Lecture 08-1: Web Scraping

DATA 503: Fundamentals of Data Engineering

Lucas P. Cordova, Ph.D.

2026-03-04

This lecture covers web scraping using the Web Scraper Chrome extension. We scrape vinyl record data from Discogs, including artist names, album titles, and ratings from detail pages using a two-level scraping approach with Element (scroll) selectors.

Table of contents

1 Web Scraping with Web Scraper

1.1 What is Web Scraping?

Web scraping is the process of automatically extracting data from websites. Instead of manually copying and pasting, we configure tools to do it for us.

Why? Because life is too short to copy 10,000 rows by hand. Your fingers will thank you.

1.1.1 Use Cases for Web Scraping

- Price monitoring (e-commerce, travel)
- Research data collection (academic datasets)
- Job posting aggregation
- Social media analysis
- Building datasets that do not exist as downloads

1.1.2 The Ethics and Legality Disclaimer

Before you go scraping the entire internet:

- Check the site's `robots.txt` file (e.g., `https://example.com/robots.txt`)
- Respect rate limits, do not hammer servers
- Check Terms of Service
- Do not scrape personal/private data
- When in doubt, ask. Or just do not do it.

We are responsible data engineers, not chaos agents.

2 Web Scraper Chrome Extension

2.1 What is Web Scraper?

Web Scraper is a free Chrome/Firefox extension that lets you scrape websites without writing code. It uses a point-and-click interface to define what data to extract.

Think of it as "I can not code a spider, but I can click on things."

Install it from:

- Chrome Web Store
- Firefox Add-ons

After installation, restart your browser (or just use new tabs).

2.1.1 Accessing Web Scraper

1. Open Chrome DevTools (`F12` or `Cmd+Opt+I` on Mac)
2. Look for the **Web Scraper** tab in DevTools
3. If you do not see it, you may need to click the `>>` arrows to find it

That is where all the magic happens.

2.1.2 Key Terminology: Sitemap

A **Sitemap** is your scraping project. It defines:

- **Start URL(s)** where the scraping begins
- **Selectors** what data to extract and how to navigate
- The overall structure of your scrape

Think of it like a treasure map, except the treasure is data and X marks the CSS selector.

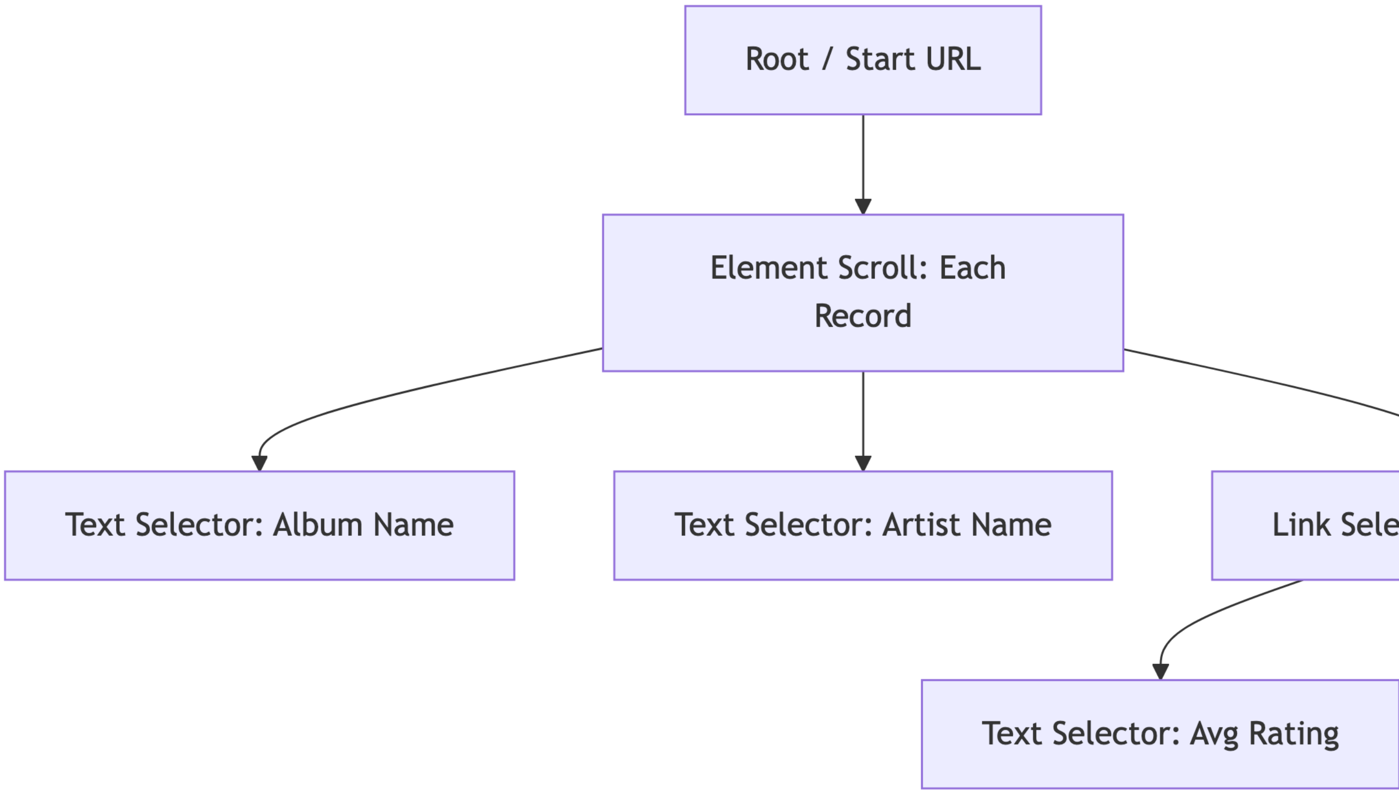
2.1.3 Key Terminology: Selectors

Selectors are the building blocks of your sitemap. Web Scraper has three categories:

Category	Purpose	Examples
Data extraction	Pull data from elements	Text, Link, Image, Table, HTML
Link navigation	Follow links to other pages	Link selector
Element grouping	Group related data together	Element selector, Element click, Element scroll

2.1.4 Key Terminology: Selector Tree

Selectors are organized in a **tree structure**. The scraper executes them top-down:


Root / Start URL
Element Scroll: Each Record
Text Selector: Album Name
Text Selector: Artist Name
Link Sele
Text Selector: Avg Rating

Parent selectors define scope. Child selectors extract data within that scope.

2.1.5 Key Terminology: Multiple Records

When a selector has **Multiple** checked, it means:

- "There are many of these on the page"
- The scraper will iterate through all matching elements
- Each match becomes a separate data row

For example: a product listing page has 25 items. The Element selector with Multiple checked will find all 25.

2.1.6 The Select Tool

Web Scraper's point-and-click tool:

1. Click **Select** in the selector creation interface
2. **Yellow** highlight = element that will be selected on click

3. **Red** highlight = already selected element
4. Click again to deselect

Keyboard shortcuts (after clicking Select):

Key	Action
`P`	Expand to parent element
`C`	Narrow to child element
`S`	Select without clicking (for dynamic elements)
`Shift`	Select multiple element groups

2.1.7 Element Preview and Data Preview

Always use these before running a scrape:

- **Element Preview** highlights which elements on the page match your selector (visual check)
- **Data Preview** shows the actual data that will be extracted (sanity check)

If the preview looks wrong, the scrape will be wrong. Trust the preview.

2.1.8 Element Scroll Selector

The **Element (scroll)** selector is used for pages that load content dynamically as you scroll (infinite scroll or lazy loading). It:

- Automatically scrolls the page to load more content
- Groups repeating elements into containers
- Works great for modern sites that load items on demand

This is the selector type we will use for Discogs search results, since the page loads items as you scroll down.

2.1.9 Range URLs for Multi-Page Scraping

Instead of relying on pagination buttons, use **range URLs**:

Pattern	Generates
`https://example.com/page/[1-3]`	`/page/1`, `/page/2`, `/page/3`
`https://example.com/page/[001-100]`	`/page/001`, `/page/002`, … (zero-padded)

Pattern	Generates
`https://example.com/page/[0-100:10]`	`/page/0`, `/page/10`, `/page/20`, …

This is how we handle multi-page scrapes without needing a pagination selector.

3 Demo: Scraping Discogs

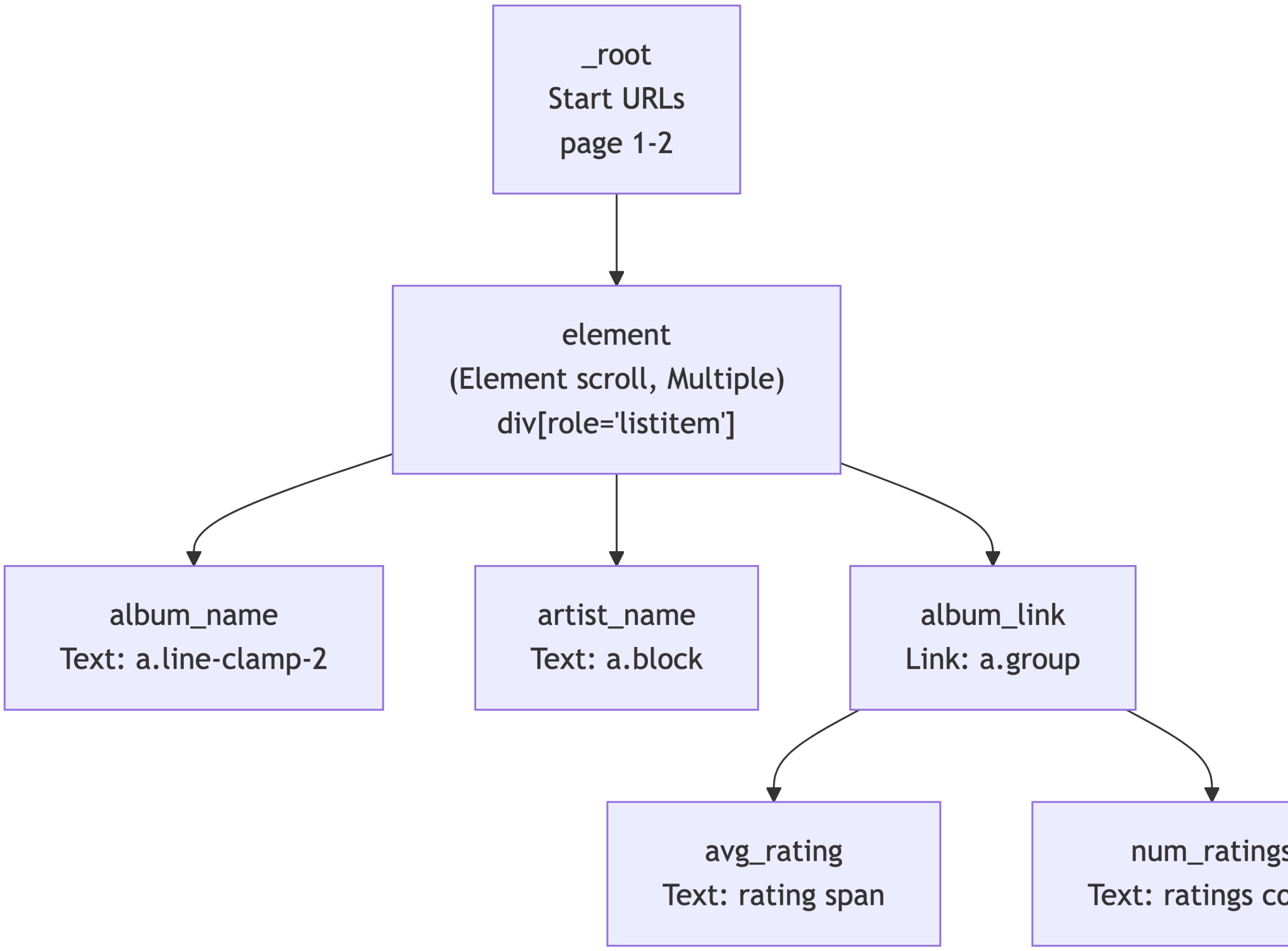
3.1 Our Target: Discogs Most Collected Releases

We are going to scrape the Most Collected Releases from Discogs, the world's largest music database.

We want: **album name**, **artist name**, and then we will **follow the link** to each album's detail page to grab the **average rating** and **number of ratings**. This is a two-level scrape.

3.1.1 The Sitemap Structure

Our scrape has two levels: the search results page and each album's detail page:



The **element** selector (Element scroll) handles grouping each search result card. Under it, we extract text (album_name, artist_name) and follow links (album_link) to detail pages where we grab ratings.

3.1.2 Step 1: Create a New Sitemap

1. Open DevTools (`F12` or `Cmd+Opt+I`) and go to the **Web Scraper** tab
2. Click **Create new sitemap** > **Create Sitemap**
3. **Sitemap name**: `discogs5`
4. **Start URL**: `https://www.discogs.com/search?sort=have%2Cdesc&type=release&page=[1-2]`

We use `[1-2]` to scrape the first 2 pages (~50 results). You can increase this later, but start small when testing.

3.1.3 Step 2: Add the Element Scroll Selector

This groups each search result card on the page and handles scroll-based loading:

1. Make sure you are at the **_root** level of selectors
2. Click **Add new selector**
3. **ID**: `element`
4. **Type**: Element scroll
5. **Selector**: `div[role='listitem']`
6. Check **Multiple** (there are many result cards per page)
7. Click **Save selector**

Why Element (scroll)? Discogs search results load dynamically. The Element scroll selector automatically scrolls to load all items and groups each result card as a container for child selectors.

3.1.4 Step 3: Add the Album Name Text Selector

Navigate **into** the `element` selector (click on it), then:

1. Click **Add new selector**
2. **ID**: `album_name`
3. **Type**: Text
4. **Selector**: Click **Select**, then click an album title in the search results. The CSS selector should be `a.line-clamp-2`
5. Leave **Multiple** unchecked (one album name per result card)
6. Click **Save selector**

Use **Data Preview** to verify it is grabbing album names correctly.

3.1.5 Step 4: Add the Artist Name Text Selector

Still inside the `element` selector:

1. Click **Add new selector**
2. **ID**: `artist_name`
3. **Type**: Text
4. **Selector**: Click **Select**, then click an artist name. The CSS selector should be `a.block`
5. Leave **Multiple** unchecked

6. Click **Save selector**

Data Preview should now show artist names alongside album names.

3.1.6 Step 5: Add the Album Link Selector

This is the key step. We need to follow the link to each album's detail page to get ratings:

1. Still inside `element`, click **Add new selector**
2. **ID**: `album_link`
3. **Type**: Link
4. **Selector**: Click **Select**, then click the album card/link area. The CSS selector should be `a.group`
5. Check **Multiple** (there are many album links per page)
6. **Link type**: linkFromHref
7. Click **Save selector**

This tells the scraper: "For each result on the page, follow this link to the detail page." The child selectors under `album_link` will extract data from those detail pages.

3.1.7 Step 6: Add Detail Page Selectors

Navigate **into** the `album_link` selector. Now we define what to extract from each album's detail page.

Average Rating:

1. Click **Add new selector**
2. **ID**: `avg_rating`
3. **Type**: Text
4. **Selector**: Navigate to any album detail page manually to build this selector. Find the average rating value. The CSS selector should be `.section_Odw8o div div ul:nth-of-type(1) span:nth-of-type(2)`
5. Click **Save selector**

Number of Ratings:

1. Click **Add new selector**
2. **ID**: `num_ratings`
3. **Type**: Text
4. **Selector**: Find the total number of ratings on the detail page. The CSS selector should be `#release-stats li:nth-of-type(4) a`
5. Click **Save selector**

3.1.8 The Complete Selector Tree

Your sitemap should now look like this:

Selector ID	Type	Parent	CSS Selector
`element`	Element scroll	`_root`	`div[role='listitem']`
`album_name`	Text	`element`	`a.line-clamp-2`
`artist_name`	Text	`element`	`a.block`
`album_link`	Link	`element`	`a.group`
`avg_rating`	Text	`album_link`	`.section_0dw8o div div ul:nth-of-type(1) span:nth-of-type(2)`
`num_ratings`	Text	`album_link`	`#release-stats li:nth-of-type(4) a`

3.1.9 Importing the Sitemap Directly

If you prefer to skip building it manually, you can import the complete sitemap JSON:

1. Go to the **Web Scraper** tab in DevTools
2. Click **Create new sitemap** > **Import Sitemap**
3. Paste this JSON and click **Import**:

```json
{
  "_id": "discogs5",
  "startUrl": [
    "https://www.discogs.com/search?sort=have%2Cdesc&type=release&page=[1-2]"
  ],
  "selectors": [
    {
      "id": "element",
      "parentSelectors": ["_root"],
      "selector": "div[role='listitem']",
      "multiple": true,
      "type": "SelectorElementScroll",
      "delay": 2000
    },
    {
      "id": "album_name",
      "multiple": false,
      "parentSelectors": ["element"],
      "selector": "a.line-clamp-2",
      "type": "SelectorText",
      "version": 2
    },
    {
      "id": "artist_name",
      "multiple": false,
      "parentSelectors": ["element"],
      "selector": "a.block",
      "type": "SelectorText",
      "version": 2
    },
    {
      "id": "album_link",
      "linkType": "linkFromHref",
      "multiple": true,
      "parentSelectors": ["element"],
      "selector": "a.group",
      "type": "SelectorLink",
      "version": 2
    },
    {
      "id": "avg_rating",
      "multiple": false,
      "parentSelectors": ["album_link"],
      "selector": ".section_Odw8o div div ul:nth-of-type(1) span:nth-of-type(2)",
      "type": "SelectorText",
      "version": 2
    },
    {
      "id": "num_ratings",
      "multiple": false,
      "parentSelectors": ["album_link"],
      "selector": "#release-stats li:nth-of-type(4) a",
      "type": "SelectorText",
      "version": 2
    }
  ]
}
```

3.1.10 Step 7: Preview and Validate

Before running the full scrape, always check:

1. Click **Element preview** on each selector. Are the correct elements highlighted?
2. Click **Data preview** on `album_name` and `artist_name`. Do they show real data?
3. Navigate to a detail page manually and test `avg_rating` and `num_ratings` element previews there

If the preview looks wrong, the scrape will be wrong. Trust the preview.

3.1.11 Step 8: Run the Scrape

1. Click **Scrape** in the Sitemap menu
2. Set **Request interval**: 2000ms (be polite to Discogs servers)
3. Set **Page load delay**: 2000ms
4. Click **Start scraping**

A popup window will open and start loading pages. It will visit each search results page, extract album/artist names, then follow each album link to grab ratings from the detail pages. Do not close the popup. Go get coffee this is a two-level scrape so it takes longer than a single-page scrape.

3.1.12 Step 9: Export the Data

Once scraping is complete:

1. Click **Browse** to preview scraped data
2. Click **Export data as CSV**
3. Save the file. This is your raw dataset!

Congratulations, you just built a two-level scraping pipeline with zero code. You scraped search results AND followed links to detail pages for extra data. Your future self who has to write Python scrapers will be jealous.

3.1.13 What We Scraped

Field	Source	Description
`album_name`	Search results page	Album/release title
`artist_name`	Search results page	Artist or band name
`album_link`	Search results page	URL to the album detail page

Field	Source	Description
`avg_rating`	Album detail page	Average user rating (e.g., 4.21)
`num_ratings`	Album detail page	Total number of user ratings

This is our **raw data**. In a real pipeline, we would clean, transform, and load this into a database.

4 In-Class Exercise: Top Vinyl of the 2010s

4.1 Your Turn: Scrape the Top 200 Vinyl Releases of 2010-2020

Work individually or with a neighbor. You are going to build your own sitemap from scratch using what we just learned.

The Use Case: Discogs tracks which releases are the most collected by users worldwide. We want to find the most collected **vinyl** releases from the 2010s decade and pull rating data from each album's detail page.

4.1.1 The Search URL

Start with this base URL in your browser:

```
https://www.discogs.com/search?sort=have%2Cdesc&type=release&year1=2010&year2=2020&format_exa
```

This applies the following filters:

Filter	Value
Sort	Most Collected (have, descending)
Type	Release
Year Range	2010 to 2020
Format	Vinyl

Open this URL in Chrome and verify you see vinyl releases sorted by collector count.

4.1.2 Your Task

Build a Web Scraper sitemap that:

1. Scrapes the first **4 pages** of results (~100 releases) using range URLs
2. Uses an **Element scroll** selector with `div[role='listitem']` to group each result
3. Extracts the **artist name** and **album name** from each search result
4. **Follows the link** to each album's detail page
5. Extracts the **average rating** and **number of ratings** from the detail page

Bonus fields (if you finish early):

- Year and format info from the search results page
- "Have" count and "Want" count from the detail page

4.1.3 Hints

- Your sitemap structure should look very similar to the `discogs5` demo we just built
- Use an **Element scroll** selector at the root level with `div[role='listitem']`
- Use a **Link** selector to navigate to detail pages
- Use the **Select** tool and **Data Preview** to verify each selector before moving on
- If a CSS selector does not work, try using `P` (parent) or `C` (child) keys while the Select tool is active

4.1.4 Expected Output

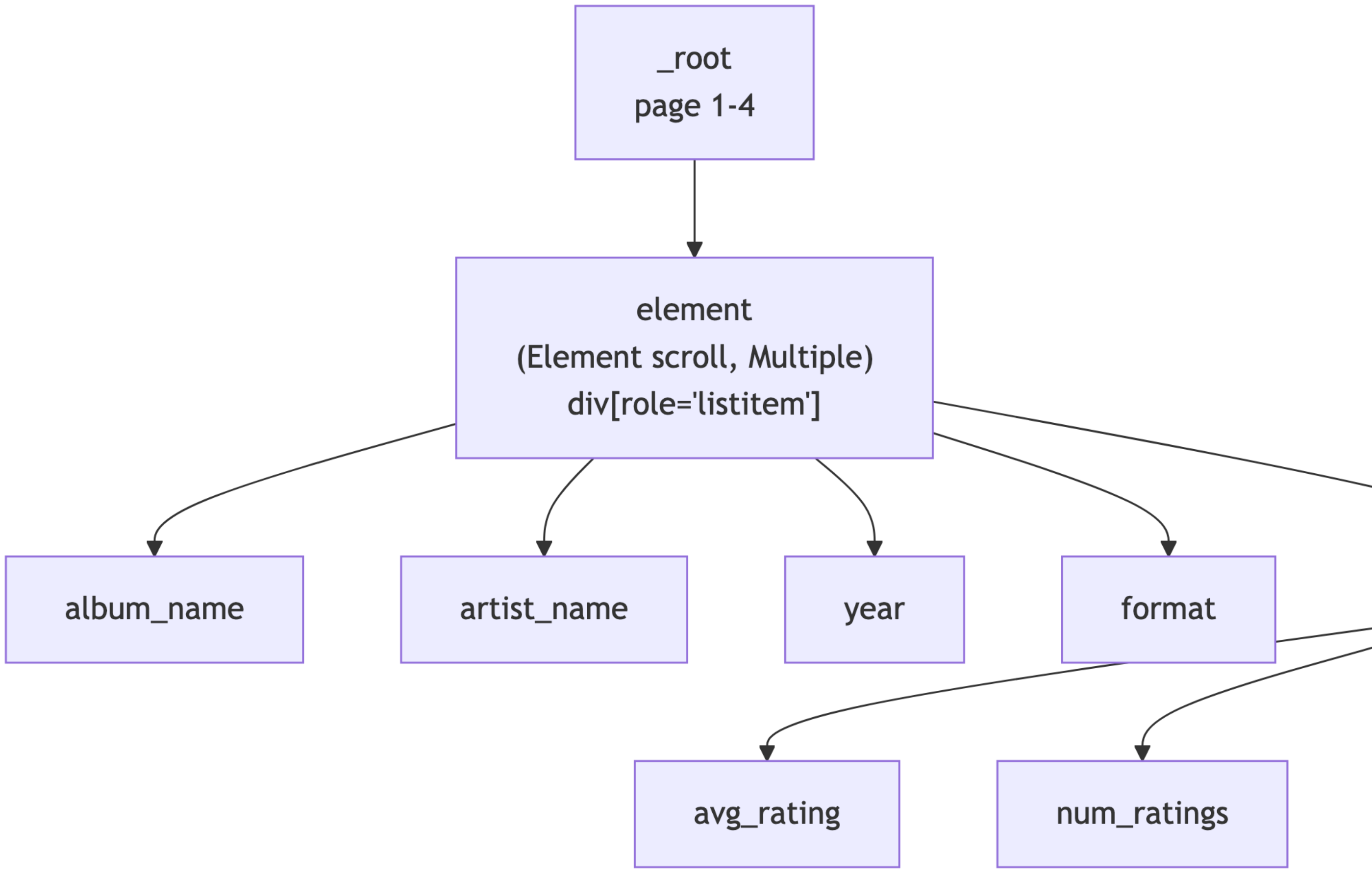
When you export to CSV, each row should have:

Column	Example Value
`artist_name`	Adele
`album_name`	21
`album_link`	https://www.discogs.com/release/...
`avg_rating`	4.18
`num_ratings`	1,247

You have **15 minutes**. When you are done (or stuck), we will compare sitemaps.

4.1.5 Solution: The Complete Sitemap

Here is a working sitemap for this exercise. You can import it via **Create new sitemap** > **Import Sitemap**:



The solution sitemap JSON is available as `top100_2010-2020.json` on the course site.

Selector ID	Type	Parent	CSS Selector
`element`	Element scroll	`_root`	`div[role='listitem']`
`album_name`	Text	`element`	`a.line-clamp-2`
`artist_name`	Text	`element`	`a.block`
`year`	Text	`element`	`span.block.text-xs`
`format`	Text	`element`	`p.text-xs.truncate`
`album_link`	Link	`element`	`a.group`

Selector ID	Type	Parent	CSS Selector
`avg_rating`	Text	`album_link`	`.section_0dw8o div div ul:nth-of-type(1) span:nth-of-type(2)`
`num_ratings`	Text	`album_link`	`#release-stats li:nth-of-type(4) a`
`num_have`	Text	`album_link`	`#release-stats li:nth-of-type(1) a`
`num_want`	Text	`album_link`	`#release-stats li:nth-of-type(2) a`

5 References

5.1 References

1. Web Scraper Documentation. https://webscraper.io/documentation
2. Discogs Search. https://www.discogs.com